Using Program Slicing Metrics for the Analysis of Code Change Processes

Raula Gaikovina Kula, Kyohei Fushida, Norihiro Yoshida, and Hajimu Iida
Graduate School of Information Science, Nara Institute of Science and Technology
Takayamacho 8916-5, Ikoma, Nara 630-0101, JAPAN
{raula-k,kyohei-f,yoshida}@is.naist.jp, iida@itc.naist.jp

ABSTRACT

Software Process Improvement is increasingly becoming a major activity for most software development organizations due to the benefits seen in the cost and business value. Software process assessments, however, can be tedious, complicated and costly. This research explores alternative assessments by analysis of fine-grained processes with its related source code characteristics.

This approach attempts to propose program slicing metrics for code changes in a software project. Using the wxWidgets software project as the case study, through project artifacts such as issue tracking system and change logs, this research explored relationships between code characteristics and its fined grained processes.

This research contributes to the development of assessment tools to assist with Software Process Improvement. It opens possibilities for assessment tools for fine-grained software processes.

Categories and Subject Descriptors

D.2.8 [**Software Engineering**]

General Terms

Experimentation

Keywords

Program Slicing, Code Change Process, Micro Process Analysis

1. INTRODUCTION

1.1 Background and Related Works

The related works are divided into two parts 1) Software Process Assessment and 2) Program Slicing.

Improvement of software processes in software development is seen as a major activity for larger software organizations as benefits are seen in the cost and business value of improvement efforts, as well as the yearly improvement in productivity of development, early defect detection and maintenance, and faster time to market [1].

A number of studies have outlined issues relating common software process improvement assessment methodologies such as CMMI[2] and ISO 9000 [3] [4] [5] [6]. Much of the issues relate to cost of assessment and implementation, especially for small software organizations [7]. In addition, these generic processes assessments are sometimes tedious and not specifically tailored. Other demotivators are higher management support, training, awareness, allocation of resources, staff involvement, experienced staff and defined software improvement process implementation methodology is critical for software process improvement [8].

There has been several related work into trying to make the models easier and better to use. Yoo [9] suggests a model that combines CMMI and ISO methods. Amburst [10] takes a different approach by treating software as manufacturing product lines, therefore making the processes systematic and generic.

An alternative measure of software processes can be done through the inspection of process execution histories. Performed at the developer's level, these measures are referred to as fine-grained processes 'micro processes'. Related research has been done by Morisaki el al [11]. This approach analyzes the quality of fine grained processes by studying the process execution histories such as change logs. This is called Micro Process Analysis (MPA). Quality of the process is measured by the execution sequence and occurrences of missing or invalid sequences.

Program Slicing is a concept first described by Weiser [12] as a method to study the behavior of source code through the flow dependency and control dependency relationship among statements. Program slicing metrics can be utilized various applications such as software inspection, which can be used to reduce defects at a fine-grained level [13]. Other related works have used program slicing metrics to classify bugs. Pan et. al. [14] showed that program slicing metrics work as well as conventional bug classification methodologies. This research proposes program slicing based metrics to describe code changes.

Combining both fields, our research aims better understand fine-grained software process by studying MPA and their code change characteristics using program slicing techniques. Our previous research with MPA and program slicing techniques [15] had focused particularly on the bug fixing process and analyses of the program slicing metrics were performed at file level. The research indicated that there is indeed a relationship between how the micro processes are executed and code-based characteristics such as McCabe's Cyclomatic Complexity (CC) [16] and lines of code (LoC).

1.2 Research Objective

Our motivation is to present a simple alternative model for software process improvement at the micro level. Current models of process assessment are at the software life cycle level and require complicated assessments with highly trained assessors. More specifically, this research is part of a proof of concept towards a prediction model based on source code properties.

Building from our previous works [16][18], this paper explores program slicing metrics specifically at *function level*, with the aim to improve from our work by only including functions edited during a code change. Also previous works focused on bug fixes and the bug fixing process, however, it was found that bug fixes, enhancements and patches are not clearly categorized. Therefore this research we broadened our scope to all code changes and the micro processes that they follow.

Based on our objective and considering code characteristics are the properties of a program slice, we formulated and tested the following research questions:

- *RQ1:* Do Code changes with similar micro process execution have similar code characteristics?
- *RQ2*: Are Program slicing metrics useful when finding a relationship between code change processes and code characteristics?

Section 2 presents the methodology as well as the proposed approach. Section 3 then explains the experiment using the approach and tools to carry out the experiments. Section 3 also presents the results of the experiments. Section 4 is a discussion and analysis of the results as well as future work. Finally, section 5 outlines the conclusion.

2. METHODOLOGY

2.1 Code Change Processes

This research focuses on the day to day processes performed in the development of software. This paper limits its data collection of code change information from the following two repositories:

1) Issue Tracking Repository

Typically a software development team uses a system to manage bugs and patches in a medium to large scale project. Trac is a wiki and issue tracking system Trac is used to track progress of any type of changes to the source code. This research utilized the Trac Management System[1] to help track all the code changes in a software project.

2)Source Code Repository

The source code repository refers the system manages changes to source code in a software project. The two main system used is the Subversion (SVN) and Concurrent Versioning System (CVS). For this research, both SVN and CVS repositories were used to gather data on bug characteristics.

[1] http://trac.edgewall.org/

To understand the micro processes behind code fixes we inspected the workflows of how fixes and changes are implemented in a software project. Inspecting the workflow of the bugs and patches in the tracking system we can be able to group and categorize code changes. Since each project has its own tailored workflow, we used workflows tailored to the project.

2.2 Characterization of Code Changes using Program Slicing Metrics.

To assess the bug characteristics, program slicing metrics are used in this approach to describe code change characteristics. This research looks to use program slicing as measure of the code changes.

In this research we refer to *editedfunction* as functions that are edited during a code change. This is the criterion of the program slice. *Back slice functions* of a program slice as functions that the edited function is dependent on. For *forward slice functions* we refer to functions that the edited function has an effect on. This is illustrated in Figure 1 below.

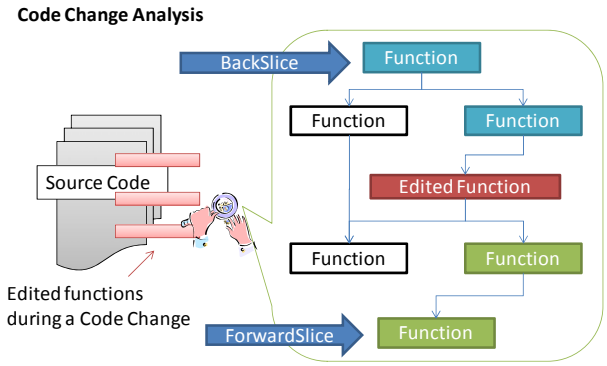

Figure 1. Example of BackSlice and ForwardSlice for a program slice

2.3 Code Change Extraction and Analysis

The analysis of the code change process includes 3 steps: 1) micro process extraction and analysis and 2) code based metrics extraction and analysis, and 3) grouping and comparing data. Details of each step are described as follows:

1) Micro process extraction and analysis:

This step involves data mining and extraction of code chance related attributes from both the source code repository and the issue tracking system. From the source code repository for each fix, the functions that were edited were extracted along with the date of edit. During the extraction, the exact affected functions are retrieved by comparing the fixed change set revision against the previous revision.

2) Code Based Metrics:

This step involves analysis of the code using the program slicing metrics. The following explains in detail how each metric was used to for bug classification. Our previous work [17] used

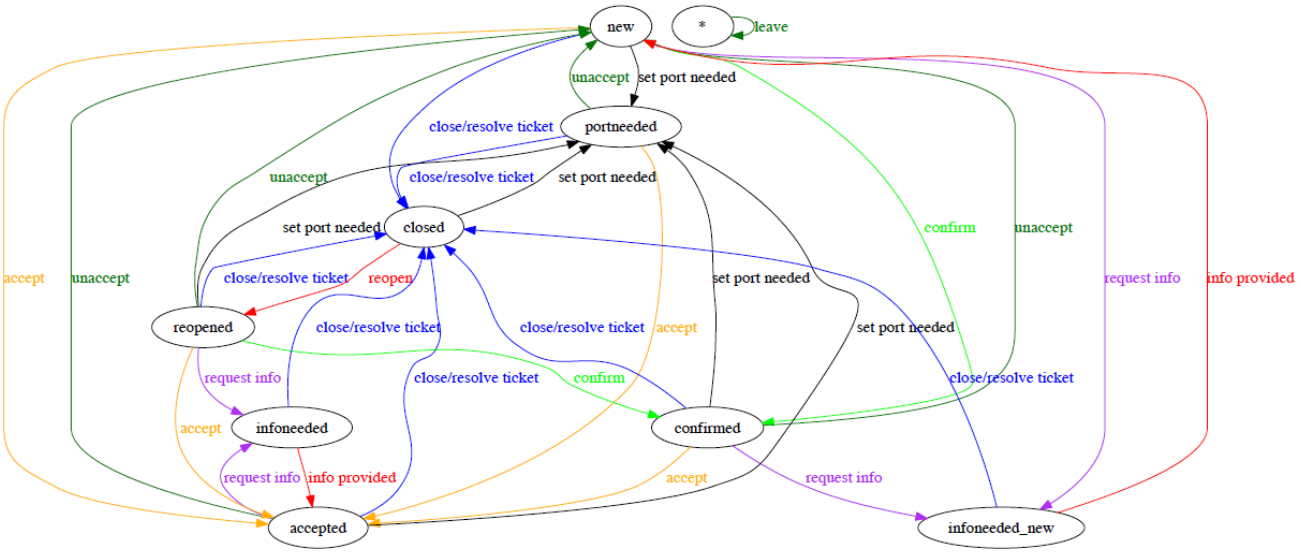

Figure 2. The micro process workflow of code change in wxWidgets (*http://trac.wxWidgets.org/wiki/wxTracWorkflow*).

Cyclomatic Complexity (CC) and Lines of code (LoC) as bug characteristic metrics. These two metrics were originally chosen as CC would be an attempt to measure the complexity of the function and LoC to measure the size of the change. In this research we use CC, however, we replaced LoC with counting the number of functions as a similar measure of size. Also we have some non-program slicing metrics to compare effectiveness.

The follow metrics were proposed and used:

- **Edited Function CC:**

This is a non-program slicing metric.If *editedfunction* is defined as a set of functions that were edited during a code change and *CC(f)* is the Cyclomatice Complexity (CC) of a function, EditedFuncCC metric is defined as:

$$EditedFuncCC = \bigcup_{f \in editedfunction} CC(f) \quad (1)$$

- **Back and Forward CC slice functions:**

Equations 2 and 3 show the calculation of how a backSliceCC and fwdSlice is calculated. As explained in Figure 1, *BackSlice* and *forwardSlice* results of when each *editedfunction* is sliced to find its depending and dependant functions. The CC for all associated functions for all *editedfunctions* are combined to give the resultant CC for the code change.

$$BackSliceCC = \bigcup_{f \in backSlice} CC(f) \quad (2)$$

$$FwdSliceCC = \bigcup_{f \in forwardSlice} CC(f) \quad (3)$$

- **Number of Edited Functions:**

$$NumEditFunc = |editedfunction| \quad (4)$$

Equation 4 states that *NumEditFunc* is the total number of edited functions. This metric is used to count how many functions are edited in a code change. This is non program slicing metric.

- **Number of Back and Forward Slice functions:**

$$BackSliceCount = \bigcup_{f \in backslice} |f| \quad (5)$$

$$FunctionSliceCount = \bigcup_{f \in forwardSlice} |f| \quad (6)$$

Equations 5 and 6 illustrate that by taking the cardinality of the *backSlice* and *forwardSlice* in relation to the *editedfunction*, we can count how many functions were associated with the code change. This metric is used to count the functions that are either depend and are dependent on a code change.

3) Groupings and comparing data:

After grouping the code changes based on their micro process execution we applied the metrics formulated in the previous step. We then analyze to find trends and relationships between micro process execution and the metrics.

3. CASE STUDY: WXWIDGETS

To test the methodology we conducted our case study using the Open Source Software wxWidgets[2] WxWidgets is a C++ library that lets developers create applications for Windows, OS X, Linux and UNIX on 32-bit and 64-bit architectures, as well as several mobile phones platforms including Windows Mobile, iPhone SDK and embedded GTK+.

The program slicing tool CodeSurfer [18] was used to generate the program slices. Our java based extraction tool acted as a web spider searching and parsing the online data repositories. Data

[2] http://www.wxWidgets.org/

was extracted from the wxWidgets Trac system and source code repository.[3]

3.1 Findings

3.1.1 Data Extraction and Selection

We analyzed wxWidgets version 2.8.11 (as of 04/16/2010) which had a size of 620389 lines of code containing 23203 functions.

Using our data extraction tool we analyzed 660662 change sets (Revision changesets 1 – 64005). The criteria for change set selection was based on the following factors 1)The code fix was linked to the wxWidgets Trac, therefore having a issue identification number and 2).The code fix contained functions that existed in the wxWidgets version that was sliced. 507 change sets passed this criteria and therefore used for analysis.

3.1.2 Micro Processes of Code Changes

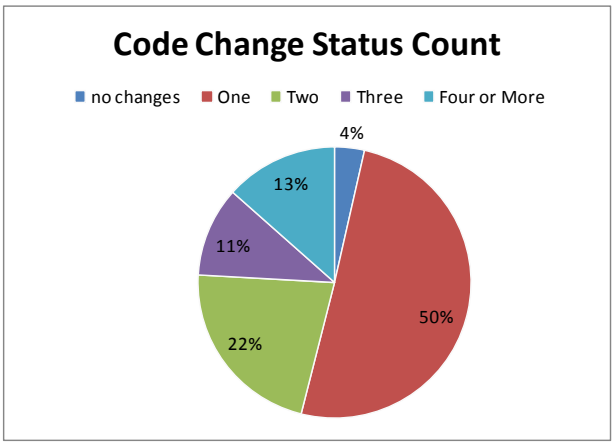

Figure 3. Status change for code change

Figure 2 illustrates the workflow used by wxWidget developers when dealing with a code change, this is regardless of being a bug or a patch or an enhancement model. Using this model we can see the states of the process of implementing a new code change. The change states are: *new, confirmed, closed, reopened, info_needed, port needed and new info_needed new*. As shown in Figure 3 using our extraction tool we were able to for each fix, find the number of status changes.

3.1.3 Grouping and comparing Code Changes

Using the grouping of the status counts as outlined in Figure 3, we then applied our proposed program slicing metrics. The results are illustrated in both Figures 4 and 5.

Figure 4 shows the proposed metrics used to compare and measure the sum of the CC of the functions affected. Starting from the left, the first graph shows the sum of the CC of the functions. However, when looking at the other two graphs, the program slice metrics for *BackSliceCC* in particular does not follow the trend as code changes with more than three changes have lower *BackSliceCC* edited. As seen, there is a trend that as the number of status changes increases with CC.

Figure 5 also suggests the correlation between the change status count and the number of functions edited. Similar to the CC metrics, it seems the number of functions increase as the number of times a status changes increase. Furthermore, with the program slicing based metrics *BackSliceCount* and *ForwardSliceCount* also support the trend.

4. DISCUSSION

4.1 Threats to Validity

Firstly the main threats to validity would be the accuracy of the data extracted as well as statistical analysis of the groupings. Still as a proof of concept, we are more interested in a correlation between program slicing metrics and micro processes. It is envisioned that once a reliable set of metrics is found, further methods of statistical tests can be applied to prove the validity of the data. We also hope to test across more different software projects.

Another threat is that the data analyzed are related to the software release version analyzed. In addition, CodeSurfer does not handle interface related code. When performing the data extraction, we only realized that out 660662 change sets, only 500 code sets were related to the version that we are analyzing. This is probably due to disappearance of functions or files or interface related code changes.

Finally code changing workflows are tailored specific therefore the status counts and complexity of micro processes will differ from organization to organization.

4.2 Evaluation

When looking at the workflow of wxWidgets in Figure 2, the workflow seems to be tedious it can be assumed that since 50% just use one state change, it does contradict the need for such a complex workflow.

Another interesting discussion point is the determination of the complexity of a code change based on its code changes. According to normal micro process analysis (MPA) [11] a normal change entails the following sequence: 1) reporting and confirmation of an issue, 2) assignment to developers, 3) Testing and applying code changes 4) closing of the issue. Findings however, suggest maybe the steps maybe be skipped. Causes could be single developer assignment and working on code change, being a simple fix or such events are not reported into the system. Further work will involve further investigation of the exact type of status change and to identify what are the 'normal' state change for a typical code change.

When applying the proposed metrics to the code changes based on their status counts we found an interesting correlation. Although not statistically proven, the number of functions increases as the change status increases. This was even evident in non-program slicing metrics of sum CC of edited functions and number of edited functions. Again further work is proposed to investigate

[3] http://trac.wxWidgets.org/ and

http://svn.wxWidgets.org/viewvc/wx/wxWidgets/

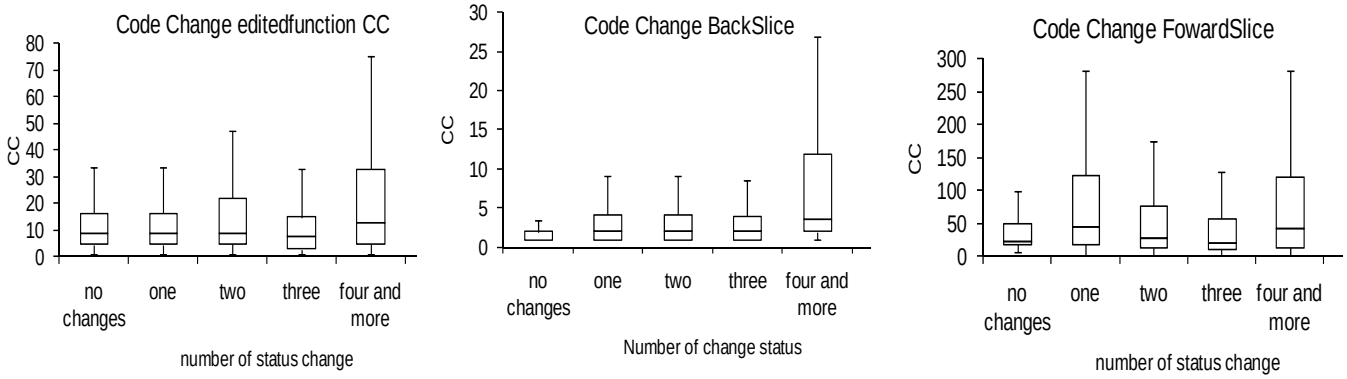

Figure 4. Graphs showing the CC of edited and program sliced functions grouped by status count of code change

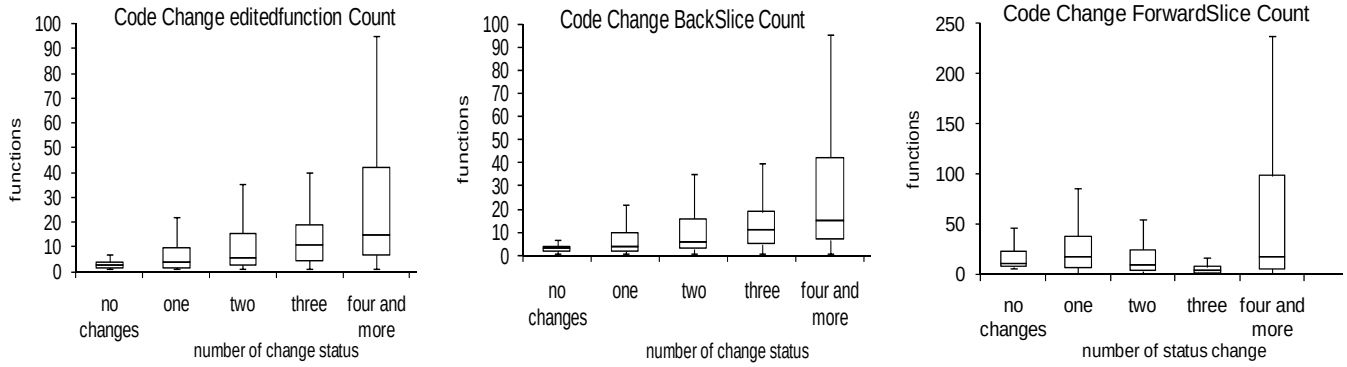

Figure 5. Graphs showing the number of edited and program sliced functions grouped by status count of code change

this further. However this correlation is not consistent with the CC metrics as seen with *ForwardSliceCC*. Again further case studies and other metrics may be proposed to better evaluate these trends.

4.3 Testing Hypothesis and Future Work

When referring to the research questions RQ1 was proven correct as we see a relationship between the complexity of the micro processes and the program slicing metrics of the affected code. Results showed that for RQ2, more current results have mixed results as *BackSliceCC* did not follow a relationship. However it may be due to being a bad metric as other program slicing metrics showed the trend. Therefore RQ2 still is not fully proven to be true.

In regards to the objective, this research contributes to the feasibility of the relationship of micro processes of code changes and their code based characteristics. This can be implemented by a framework that will help assist developers identify code changes that have a high likelyhood of having complicated micro processes.

This work is showing promise as it agrees with previous work that there is a correlation between code-based characteristics and MPA.

Further investigation will be performed to find out the exact state changes. Moreover, formulating and testing additional program slicing metrics. Once usable metrics are established, other software projects can be used as case studies.

5. CONCLUSION

The main objective of the research is to provide a proof of concept that there is indeed a relationship between a code change attributes and how its related processes are executed. The results suggest there is a correlation between how much of the complexity of the micro process and its program slicing metrics. Further work is proposed to refine the metrics as well as test on other similar software projects.

It is envisioned that the research will contribute to a better understanding and classification of code changes based on the nature of code, therefore using the nature of the code to suggest improvement of software processes at the micro level.

ACKNOWLEDGMENTS

Special thanks to Shinji Kawaguchi for his contribution to this research in MPA. This work is being conducted as a part of the StagE project, The Development of Next-Generation IT Infrastructure, supported by the Ministry of Education, Culture,

Sports, Science and Technology. This research was also supported by Japan Society for the Promotion of Science, Grant-in-Aid for Scientific Research (No.22500027), and Grant-in-Aid for Research Activity start-up (No.22800040 and 22800043).

REFERENCES

[1]. J. Herbsleb, A. Carleton, J. Rozum, J. Siegel and D. Zubrow, "Benefits of CMM-Based Software Process Improvement: Initial Results", (CMU/SEI-94-TR-13). Pittsburgh, Pa.: Software Engineering Institute, Carnegie Mellon University, 1994.

[2]. K. K. Margaret and J. A. Kent, "Interpreting the CMMI: A Process Improvement Approach", Auerbach Publications, (CSUE Body of Knowledge area: Software Quality Management), 2003.

[3]. C. H. Schmauch, "ISO 9000 for Software Developers", 2nd. ASQ Quality Press, 1995.

[4]. S. Beecham, T. Hall and A. Rainer, "Software process problems in twelve software companies: an empirical analysis", Empirical Software Engineering, v.8, pp. 7-42, 2003.

[5]. N. Baddoo and T. Hall, "De-Motivators of software process improvement: an analysis of practitioner's views", Journal of Systems and Software, v.66, n.1, pp. 23-33, 2003.

[6]. M. Niazi, M. A. Babar, "De-motivators for software process improvement: an analysis of Vietnamese practitioners' views", in Product Focused Software Process Improvement PROFES 2007, LNCS, v.4589, pp. 118-131, 2007.

[7]. J.G. Brodman and D.L. Johnson, "What small businesses and small organizations say about the CMMI", In Proceedings of the 16th International Conference on Software Engineering (ICSE), IEEE Computer Society, pp. 331 – 340, 1994

[8]. A. Rainer and T. Hall, "Key success factors for implementing software process improvement: a maturity-based analysis", Journal of Systems and Software v.62, n.2, pp.71-84, 2002.

[9]. C. Yoo, J. Yoon, B. Lee, C. Lee, J. Lee, S. Hyun and C. Wu, "A unified model for the implementation of both ISO 9001:2000 and CMMI by ISO-certified organizations", The Journal of Systems and Software 79, n.7, pp. 954-961, 2006.

[10]. O. Armbrust, M. Katahira, Y. Miyamoto, J. Münch, H. Nakao, and A. O. Campo, "Scoping software process lines", Softw. Process , v.14, n.3, pp.181-197, May, 2009.

[11]. S. Morisaki and H. Iida, "Fine-Grained Software Process Analysis to Ongoing Distributed Software Development", 1st Workshop on Measurement-based Cockpits for Distributed Software and Systems Engineering Projects (SOFTPIT 2007), pp.26-30, Munich, Germany, Aug., 2007.

[12] M. Weiser, "Program slicing", In Proceedings of the 5th International Conference on Software Engineering. (ICSE). IEEE Press, Piscataway, NJ, pp.439-449, Mar. 09 - 12, 1981

[13]. P. Anderson, T. Reps, T. Teitelbaum, M. Zarins, "Tool Support for Fine-Grained Software Inspection," IEEE Software, pp. 42-50, July/August, 2003

[14]. K. Pan, S. Kim, E. J. Whitehead, Jr., "Bug Classification Using Program Slicing Metrics", Source Code Analysis and Manipulation, IEEE International Workshop, pp.31-42, 2006

[15]. R. G. Kula, K. Fushida, S. Kawaguchi, and H. Iida, “Analysis of Bug Fixing Processes Using Program Slicing Metrics," in Product-Focused Software Process Improvement PROFES 2010, vol. LNCS 6156, pp. 032-046, June 2010.

[16]. A. Watson and T. McCabe, "Structured testing: A testing methodology using the cyclomatic complexity metric", National Institute of Standards and Technology, Gaithersburg, MD, (NIST) Special Publication, pp.500-235, 1996.

[17]. R. G. Kula, ``Using Program Slicing Metrics for the Analysis of Bug Fixing Processes," Master thesis, Institute of Science and Technology, Nara, Japan, 2010.

[18]. P. Anderson , M. Zarins, "The CodeSurfer Software Understanding Platform", Proceedings of the 13th International Workshop on Program Comprehension, pp.147-148, May 15-16, 2005